

Web Application Description Language (WADL)

Marc J. Hadley, Sun Microsystems Inc.

November 16, 2005

Abstract

This specification describes the Web Application Description Language (WADL). An increasing number of Web based enterprises (Google, Yahoo, Amazon, Flickr to name but a few) are developing HTTP-based applications that provide access to their internal data using XML. Typically these applications are described using a combination of textual protocol descriptions combined with XML schema-based data format descriptions, WADL is designed to provide a machine process-able protocol description format for use with such HTTP-based Web applications, especially those using XML.

1 Introduction

This specification describes the Web Application Description Language (WADL). WADL is designed to provide a machine process-able protocol description format for use with HTTP-based Web applications, especially those using XML.

The following listing shows an example of a WADL description for the Yahoo News Search[1] application.

```
1 <?xml version="1.0"?>
2 <application xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3   xsi:schemaLocation="http://research.sun.com/wadl wadl.xsd"
4   xmlns:tns="urn:yahoo:yn"
5   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
6   xmlns:yn="urn:yahoo:yn"
7   xmlns:ya="urn:yahoo:api"
8   xmlns="http://research.sun.com/wadl">
9   <grammars>
10    <include
11      href="NewsSearchResponse.xsd"/>
12    <include
13      href="http://api.search.yahoo.com/Api/V1/error.xsd"/>
14  </grammars>
15
16  <resources base="http://api.search.yahoo.com/NewsSearchService/V1/">
```

```

17     <resource uri="newsSearch">
18         <method href="#NewsSearch"/>
19     </resource>
20 </resources>
21
22 <method name="GET" id="NewsSearch">
23     <request>
24         <query_variable name="appid" type="xsd:string" required="true"/>
25         <query_variable name="query" type="xsd:string" required="true"/>
26         <query_variable name="type" type="xsd:string"/>
27         <query_variable name="results" type="xsd:int"/>
28         <query_variable name="start" type="xsd:int"/>
29         <query_variable name="sort" type="xsd:string"/>
30         <query_variable name="language" type="xsd:string"/>
31     </request>
32     <response>
33         <representation mediaType="application/xml" element="yn:ResultSet"/>
34         <fault id="SearchError" status="400" mediaType="application/xml"
35             element="ya:Error"/>
36     </response>
37 </method>
38 </application>

```

Lines 2–8 begin an application description and define the XML namespaces used elsewhere in the service description. Lines 9–14 define the XML grammars used by the service, in this case two W3C XML Schema files are included by reference. Lines 16–20 describe the Yahoo News Search Web resource and the HTTP methods it supports. Lines 22–37 describe the ‘NewsSearch’ GET method: lines 23–31 describe the input; lines 32–36 describe the possible outputs.

2 Description Components

All WADL elements have the following XML namespace name:

- <http://research.sun.com/wadl>

This section describes each component of a WADL document in detail. Note that most elements are extensible either via additional attributes or child elements from another namespace. Unrecognized extensions may be ignored.

2.1 Application

The `application` element forms the root of a WADL description and contains the following:

1. An optional `grammars` element – see section 2.2.
2. An optional `resources` element – see section 2.3.
3. Zero or more of the following:
 - A `method` element – see section 2.5.
 - A `representation` element – see section 2.6.
 - A `fault` element – see section 2.7.

2.2 Grammars

The `grammars` element acts as a container for definitions of any XML structures exchanged during execution of the protocol described by the WADL document. Such definitions may be included inline or by reference using the `include` element (see section 2.2.1). No particular schema language is mandated; sections 3 and 4 describe use of RelaxNG and W3C XML Schema with WADL respectively.

It is permissible to include multiple definitions of a particular XML structure, such definitions are assumed to be equivalent and consumers of a WADL description are free to choose amongst the alternatives or even combine them if they support that capability.

2.2.1 Include

The `include` element allows the definitions of one or more XML structures to be included by reference. The `href` attribute specifies a URI for the referenced definitions and is of type `xsd:anyURI`. Use of the `include` element is logically equivalent to in-lining the referenced document within the WADL `grammars` element.

2.3 Resources

The `resources` element acts as a container for the resources provided by the application. A `resources` element has a `base` attribute of type `xsd:anyURI` that specifies the base URI for each child resource identifier (see section 2.4). Child `resource` elements describe a single resource provided by the application.

2.4 Resource

The `resource` element describes a single resource provided by the Web application. A resource is identified by a URI and the `resources` parent element (see section 2.3) specifies the base URI for all child `resource` elements. A resource element has the following attributes;

uri An optional attribute of type `xsd:anyURI`. If present, it specifies the identifier of the resource as a static, relative URI whose base URI is given by the parent `resources` element. An implicit leading `'/'` character is added to the value of the `uri` attribute. If the `uri` attribute is omitted then the `resource` element **MUST** contain a child `path_variable` element (see section 2.4.1) that defines a parameterized relative URI whose base URI is given by the parent element.

A `resource` element contains the following child elements:

- An optional `path_variable` element, see section 2.4.1 that defines a parameterized relative URI for its parent `resource` element.
- Zero or more `method` (see section 2.5) elements, each of which describes the input to and output from a HTTP protocol method that can be applied to the resource.
- Zero or more `resource` elements that describe sub-resources.

The following example shows an extract from a Web application description that provides multiple resources:

```
1 <resources base="http://example.com/widgets">
2   <resource uri="stockreport">
3     ...
4   </resource>
5   <resource>
6     <path_variable name="widgetId" type="xsd:NMTOKEN" />
7     ...
8   </resource>
9 </resources>
```

In the above example there are multiple resources:

- A single resource identified by a static URI: `http://example.com/widgets/stockreport`.
- Multiple resources identified by generative URIs: `http://example.com/widgets/widgetId`, where the `widgetId` component of the URI is replaced at runtime with the value of a runtime variable called `widgetId`.

In addition to being children of `resources` elements, `resource` elements may also be nested to an arbitrary level. In this case the child `resource` element describes a sub-resource and is identified relative to that of the parent `resource` element. The following example shows a WADL fragment that describes three resources:

```
1 <resources base="http://example.com/">
```

```

2   <resource uri="widgets">
3     ...
4   <resource uri="stockreport">
5     ...
6     <resource uri="">
7       ...
8     </resource>
9     ...
10  </resource>
11  ...
12  </resource>
13 </resources>

```

The resources described above are identified by three URIs:

- `http://example.com/widgets`
- `http://example.com/widgets/stockreport`
- `http://example.com/widgets/stockreport/`

2.4.1 Path Variable

The `path_variable` element is used to parameterize the identifier of its parent `resource` element (see section 2.4). A `path_variable` element has no defined child elements and has the following attributes:

name Specifies the name of the variable as an `xsd:NMTOKEN`. Required.

type Optionally specifies the type of the variable as an XML qualified name, defaults to `xsd:string`.

As is the case for the `uri` attribute of the `resource` element, an implicit leading `'/` character is added to the value of the path variable.

2.5 Method

A `method` element describes the input to and output from a HTTP protocol method that may be applied to a resource. A `method` element has one of the two following combinations of attributes:

1. **name** Specifies the HTTP method used.
id Specifies the identifier of the method.

2. **href** Allows a method defined elsewhere to be referenced by its `id` attribute, using a URI reference. This form of the method element is used to prevent duplication when a method defined elsewhere also applies to the parent `resource` element. When the `href` attribute is present, the method element **MUST NOT** have any child content.

It is permissible to have multiple child `method` elements that have the same value of the `name` attribute for a given resource, such siblings represent distinct variations of the same HTTP method and will typically have different input data.

A `method` element has two child elements:

request Specifies the input to the method as a collection of variables and an optional resource representation – see section 2.5.1.

response Specifies the output of the method as a collection of alternate resource representations – see section 2.5.3.

2.5.1 Request

A `request` element describes the input that may be included when applying a HTTP method to a resource. A `request` element has no attributes and may contains the following child elements in order:

1. Zero or more `representation` elements – see section 2.6. Note that use of `representation` elements is confined to HTTP methods that accept an entity body in the request (e.g. PUT or POST). Sibling `representation` elements represent logically equivalent alternatives, e.g. a particular resource might support multiple XML grammars for a particular request.
2. Zero or more `query_variable` elements – see section 2.5.2.

2.5.2 Query Variable

A `query_variable` element represents a URI query parameter as described in section 17.13 of HTML 4.01[2]. The runtime values of query variables are sent as URI query parameters when the HTTP method is invoked. A `query_variable` element has no defined child elements and has the following attributes:

name Specifies the name of the variable as an `xsd:NMTOKEN`. Required.

type Optionally specifies the type of the variable as an XML qualified name, defaults to `xsd:string`.

required Optionally specifies whether the variable is required to be present or not, defaults to false (not required).

repeating Optionally specifies whether the variable is single valued or may have multiple values, defaults to false (variable is single valued).

fixed Optionally specifies a fixed value for the variable.

The following example shows a resource with a generative URI that supports a single HTTP method with a single query variable:

```
1 <resources base="http://example.com/widgets">
2   <resource>
3     <path_variable name="widgetId" type="xsd:string"/>
4     <method id="GetDescription" name="GET">
5       <request>
6         <query_variable name="verbose" type="xsd:boolean"/>
7       </request>
8       <response>
9         ...
10      </response>
11    </method>
12  </resource>
13 </resources>
```

If the value of the `widgetId` variable is '1234567890' and the value of the `verbose` variable is 'true' then the URI on which the HTTP GET will be performed is:

```
http://example.com/widgets/1234567890?verbose=true
```

2.5.3 Response

A `response` element describes the output of performing a HTTP method on a resource. It contains zero or more of the following:

- A `representation` element - see section 2.6.
- A `fault` element - see section 2.7.

Each child `representation` element describes a resource state representation that may result from performing the method. Sibling `representation` elements indicate logically equivalent alternatives, normal HTTP mechanisms may be used to select a particular alternative. Each child `fault` element describes a fault condition that may occur – note that not all possible fault conditions are likely to be described and client applications should be prepared to handle the full range of possible HTTP error conditions.

2.6 Representation

A `representation` element describes a representation of a resource's state and can either be declared globally as a child of the `application` element, embedded locally as a child of a `request` or `response` element, or referenced externally. A `representation` element has one of the two following combinations of attributes:

1. **id** Specifies the identifier of the representation, required for globally defined representations, not allowed on locally embedded representations. Representations are identified by an XML ID and are referred to using a URI reference.

mediaType Specifies the media type of the representation.

element For XML-based representations, specifies the qualified name of the root element as described within the `grammars` section – see section 2.2.

2. **href** Allows a representation defined elsewhere to be referenced by its `id` attribute, using a URI reference. This form of the `representation` element is used to prevent duplication when a representation defined elsewhere also applies to the parent `request` or `response` element. When the `href` attribute is present, the `representation` element **MUST NOT** have any child content.

In addition to the attributes listed above, a `representation` element can have zero or more child `representation_variable` elements, see section 2.6.1.

2.6.1 Representation Variable

A `representation_variable` element is used to parameterize its parent `representation` element. A `representation_variable` element has no defined child elements and has the following attributes:

name Specifies the name of the variable as an `xsd:NMTOKEN`. Required.

type Optionally specifies the type of the variable as an XML qualified name, defaults to `xsd:string`.

path Optionally specifies a path to the value of the variable.

required Optionally specifies whether the variable is required to be present or not, defaults to false (variable not required).

repeating Optionally specifies whether the variable is single valued or may have multiple values, defaults to false (variable is single valued).

fixed Optionally specifies a fixed value for the variable.

Representation variables can have one of two different functions depending on the media type of the representation:

1. Define the content of the representation. For `representation` elements with a `mediaType` attribute whose value is either `'application/x-www-form-urlencoded'` or `'multipart/form-data'` the representation variables define the content of the representation which is formatted according to the media type. The same may apply to other media types.
2. Provide a hint to processors about items of interest within a representation. For XML based representations, the representation variables can be used to pick out items of interest with the XML. When used this way they are most useful for simple representations as in the example below where the information is contained within a few well defined locations. The `path` attribute of a representation variable specifies the path to the value of the variable within the representation. For XML-based representations this is an XPath expression.

The following example shows an XML-based resource representation and two possible corresponding WADL representation elements:

```

1  <!-- XML-based representation of a widget -->
2  <w:widget xmlns:w="http://example.com/widgets">
3    <w:name>A Widget</w:name>
4    <w:description>A very useful gizmo.</w:description>
5    <w:price currency="USD">19.99</w:price>
6  </w:widget>
7
8  <!-- WADL fragment describing the widget representation without variables-->
9  <wadl:representation mediaType="application/xml" element="w:widget"/>
10
11 <!-- WADL fragment describing the widget representation with variables -->
12 <wadl:representation mediaType="application/xml" element="w:widget">
13   <wadl:representation_variable name="name"
14     type="xsd:string" path="/w:widget/w:name"/>
15   < wadl:representation_variable name="description"
16     type="xsd:string" path="/w:widget/w:description"/>
17   < wadl:representation_variable name="price"
18     type="xsd:decimal" path="/w:widget/w:price"/>
19   < wadl:representation_variable name="currency"
20     type="xsd:NMTOKEN" path="/w:widget/w:price/@currency"/>
21 </wadl:representation>

```

2.7 Fault

A `fault` element is similar to a `representation` element (see section 2.6) in structure but differs in that it denotes an error condition. A `fault` element has the same attributes as a `representation` element but may also have an additional `status` attribute that specifies a list of HTTP status codes associated with a particular error condition. Note that multiple `fault` elements may share one or more HTTP status codes, such elements may describe more granular fault conditions or may provide equivalent information in different formats.

2.7.1 Fault Variables

Fault variables are `representation_variable` elements that are direct children of a `fault` element. Fault variables perform the same function for `fault` elements that `representation` variables (see section 2.6.1) perform for `representation` elements.

3 Use of RelaxNG With WADL

One or more legal RelaxNG schemas may be embedded within a WADL `grammars` element or may be included by reference using a `include` element. Multiple RelaxNG schemas may be combined within a single schema using the facilities provided by RelaxNG (e.g. `rng:include`). The default namespace for an included RelaxNG grammar is the default namespace of the WADL `grammars` element.

The `element` attribute of `representation` and `fault` elements refers to a corresponding RelaxNG element pattern using the XML qualified name of the element.

4 Use of W3C XML Schema With WADL

One or more legal W3C XML Schemas may be embedded within a WADL `grammars` element or may be included by reference using a `include` element. Multiple W3C XML Schemas may be combined within a single schema using the facilities provided by W3C XML Schema (e.g. `xsd:include`).

The `element` attribute of `representation` and `fault` elements refers to a corresponding W3C XML Schema global element declaration using the XML qualified name of the element.

5 RelaxNG Schema for WADL

```
1 namespace a = "http://relaxng.org/ns/compatibility/annotations/1.0"
2 namespace local = ""
3 namespace wadl = "http://research.sun.com/wadl"
4
5 start =
6   element wadl:application {
7     grammars?,
8     resources?,
9     (method | representation | fault)*,
10    foreign-attribute,
11    foreign-element
12  }
13 grammars = element wadl:grammars { \include*, foreign-element }
14 resources =
```

```

15     element wadl:resources {
16         resource+,
17         attribute base { xsd:anyURI },
18         foreign-element
19     }
20 resource =
21     element wadl:resource {
22         (attribute uri { xsd:anyURI }
23         | path_variable),
24         (method | resource)+
25     }
26 method =
27     element wadl:method {
28         (attribute href { xsd:anyURI }
29         | (request?,
30         response?,
31         attribute id { xsd:token },
32         attribute name {
33             "DELETE" | "GET" | "HEAD" | "POST" | "PUT" | xsd:token ""
34         })),
35         foreign-element,
36         foreign-attribute
37     }
38 request =
39     element wadl:request {
40         representation*, query_variable*, foreign-attribute, foreign-element
41     }
42 response =
43     element wadl:response {
44         (representation | fault)*, foreign-attribute, foreign-element
45     }
46 representation =
47     element wadl:representation {
48         (attribute href { xsd:anyURI }
49         | (representation_variable*,
50         attribute id { xsd:token }?,
51         attribute element { xsd:QName }?,
52         attribute mediaType { text }?)),
53         foreign-attribute,
54         foreign-element
55     }
56 fault =
57     element wadl:fault {
58         (attribute href { xsd:anyURI }
59         | (representation_variable*,
60         attribute id { xsd:token }?,
61         attribute element { xsd:QName }?,
62         attribute mediaType { text }?,
63         attribute status {
64             list { xsd:int+ }
65         }?)),
66         foreign-attribute,
67         foreign-element
68     }

```

```

69 path_variable =
70   element wadl:path_variable {
71     attribute name { xsd:token },
72     attribute type { text }?,
73     foreign-element,
74     foreign-attribute
75   }
76 query_variable =
77   element wadl:query_variable {
78     attribute name { xsd:token },
79     attribute type { text }?,
80     attribute required { xsd:boolean }?,
81     attribute repeating { xsd:boolean }?,
82     attribute fixed { text }?,
83     foreign-element,
84     foreign-attribute
85   }
86 representation_variable =
87   element wadl:representation_variable {
88     attribute name { xsd:token },
89     attribute type { text }?,
90     attribute path { text }?,
91     attribute required { xsd:boolean }?,
92     attribute repeating { xsd:boolean }?,
93     attribute fixed { text }?,
94     foreign-element,
95     foreign-attribute
96   }
97 \include =
98   element wadl:include {
99     attribute href { xsd:anyURI },
100    foreign-attribute
101  }
102 foreign-attribute = attribute * - (wadl:* | local:*) { text }*
103 foreign-element =
104   element * - (wadl:* | local:*) {
105     (attribute * { text }
106     | text
107     | any-element)*
108   }*
109 any-element =
110   element * {
111     (attribute * { text }
112     | text
113     | any-element)*
114   }*

```

6 XML Schema for WADL

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"

```

```

3 targetNamespace="http://research.sun.com/wadl"
4 xmlns:tns="http://research.sun.com/wadl"
5 elementFormDefault="qualified">
6
7 <xs:element name="application">
8   <xs:complexType>
9     <xs:sequence>
10      <xs:element ref="tns:grammars" minOccurs="0"/>
11      <xs:element ref="tns:resources" minOccurs="0"/>
12      <xs:choice minOccurs="0" maxOccurs="unbounded">
13        <xs:element ref="tns:method"/>
14        <xs:element ref="tns:representation"/>
15        <xs:element ref="tns:fault"/>
16      </xs:choice>
17      <xs:any namespace="##other" processContents="lax"
18        minOccurs="0" maxOccurs="unbounded"/>
19    </xs:sequence>
20  </xs:complexType>
21 </xs:element>
22
23 <xs:element name="grammars">
24   <xs:complexType>
25     <xs:sequence>
26      <xs:element minOccurs="0" maxOccurs="unbounded" ref="tns:include"/>
27      <xs:any namespace="##other" processContents="lax"
28        minOccurs="0" maxOccurs="unbounded"/>
29    </xs:sequence>
30  </xs:complexType>
31 </xs:element>
32
33 <xs:element name="resources">
34   <xs:complexType>
35     <xs:sequence>
36      <xs:element ref="tns:resource" maxOccurs="unbounded"/>
37      <xs:any namespace="##other" processContents="lax"
38        minOccurs="0" maxOccurs="unbounded"/>
39    </xs:sequence>
40    <xs:attribute name="base" type="xs:anyURI"/>
41  </xs:complexType>
42 </xs:element>
43
44 <xs:element name="resource">
45   <xs:complexType>
46     <xs:sequence>
47      <xs:element ref="tns:path_variable" minOccurs="0"/>
48      <xs:choice maxOccurs="unbounded">
49        <xs:element ref="tns:method"/>
50        <xs:element ref="tns:resource"/>
51      </xs:choice>
52      <xs:any minOccurs="0" maxOccurs="unbounded" namespace="##other" processContents="
53    </xs:sequence>
54    <xs:attribute name="uri" type="xs:anyURI"/>
55    <xs:anyAttribute namespace="##other" processContents="lax"/>
56  </xs:complexType>

```

```

57     </xs:element>
58
59     <xs:element name="method">
60         <xs:complexType>
61             <xs:sequence>
62                 <xs:element ref="tns:request" minOccurs="0"/>
63                 <xs:element ref="tns:response" minOccurs="0"/>
64                 <xs:any namespace="##other" processContents="lax"
65                     minOccurs="0" maxOccurs="unbounded"/>
66             </xs:sequence>
67             <xs:attribute name="id" type="xs:ID"/>
68             <xs:attribute name="name" type="tns:Method"/>
69             <xs:attribute name="href" type="xs:anyURI"/>
70             <xs:anyAttribute namespace="##other" processContents="lax"/>
71         </xs:complexType>
72     </xs:element>
73
74     <xs:simpleType name="Method">
75         <xs:union memberTypes="tns:HTTPMethods xs:NMTOKEN"/>
76     </xs:simpleType>
77
78     <xs:simpleType name="HTTPMethods">
79         <xs:restriction base="xs:NMTOKEN">
80             <xs:enumeration value="GET"/>
81             <xs:enumeration value="POST"/>
82             <xs:enumeration value="PUT"/>
83             <xs:enumeration value="HEAD"/>
84             <xs:enumeration value="DELETE"/>
85         </xs:restriction>
86     </xs:simpleType>
87
88     <xs:element name="include">
89         <xs:complexType>
90             <xs:attribute name="href" type="xs:anyURI"/>
91             <xs:anyAttribute namespace="##other" processContents="lax"/>
92         </xs:complexType>
93     </xs:element>
94
95     <xs:element name="request">
96         <xs:complexType>
97             <xs:sequence>
98                 <xs:element ref="tns:representation" minOccurs="0"
99                     maxOccurs="unbounded"/>
100                <xs:element ref="tns:query_variable" minOccurs="0"
101                    maxOccurs="unbounded"/>
102                <xs:any namespace="##other" processContents="lax"
103                    minOccurs="0" maxOccurs="unbounded"/>
104            </xs:sequence>
105            <xs:anyAttribute namespace="##other" processContents="lax"/>
106        </xs:complexType>
107    </xs:element>
108
109    <xs:element name="response">
110        <xs:complexType>

```

```

111     <xs:sequence>
112         <xs:choice minOccurs="0" maxOccurs="unbounded">
113             <xs:element ref="tns:representation"/>
114             <xs:element ref="tns:fault"/>
115         </xs:choice>
116         <xs:any namespace="##other" processContents="lax"
117             minOccurs="0" maxOccurs="unbounded"/>
118     </xs:sequence>
119     <xs:anyAttribute namespace="##other" processContents="lax"/>
120 </xs:complexType>
121 </xs:element>
122
123 <xs:element name="representation">
124     <xs:complexType>
125         <xs:sequence>
126             <xs:element ref="tns:representation_variable" minOccurs="0"
127                 maxOccurs="unbounded"/>
128             <xs:any namespace="##other" processContents="lax"
129                 minOccurs="0" maxOccurs="unbounded"/>
130         </xs:sequence>
131         <xs:attribute name="id" type="xs:ID"/>
132         <xs:attribute name="element" type="xs:QName"/>
133         <xs:attribute name="mediaType" type="xs:string"/>
134         <xs:attribute name="href" type="xs:anyURI"/>
135         <xs:anyAttribute namespace="##other" processContents="lax"/>
136     </xs:complexType>
137 </xs:element>
138
139 <xs:simpleType name="faultCodeList">
140     <xs:list itemType="xs:unsignedInt"/>
141 </xs:simpleType>
142
143 <xs:element name="fault">
144     <xs:complexType>
145         <xs:sequence>
146             <xs:element ref="tns:representation_variable" minOccurs="0"
147                 maxOccurs="unbounded"/>
148             <xs:any namespace="##other" processContents="lax"
149                 minOccurs="0" maxOccurs="unbounded"/>
150         </xs:sequence>
151         <xs:attribute name="id" type="xs:ID" use="required"/>
152         <xs:attribute name="element" type="xs:QName"/>
153         <xs:attribute name="status" type="tns:faultCodeList"/>
154         <xs:attribute name="mediaType" type="xs:string"/>
155         <xs:attribute name="href" type="xs:anyURI"/>
156         <xs:anyAttribute namespace="##other" processContents="lax"/>
157     </xs:complexType>
158 </xs:element>
159
160 <xs:element name="query_variable">
161     <xs:complexType>
162         <xs:sequence>
163             <xs:any namespace="##other" processContents="lax"
164                 minOccurs="0" maxOccurs="unbounded"/>

```

```

165     </xs:sequence>
166     <xs:attribute name="name" type="xs:NMTOKEN" use="required"/>
167     <xs:attribute name="type" type="xs:QName" default="xs:string"/>
168     <xs:attribute name="required" type="xs:boolean" default="false"/>
169     <xs:attribute name="repeating" type="xs:boolean" default="false"/>
170     <xs:attribute name="fixed" type="xs:string"/>
171     <xs:anyAttribute namespace="##other" processContents="lax"/>
172   </xs:complexType>
173 </xs:element>
174
175 <xs:element name="path_variable">
176   <xs:complexType>
177     <xs:sequence>
178       <xs:any namespace="##other" processContents="lax"
179         minOccurs="0" maxOccurs="unbounded"/>
180     </xs:sequence>
181     <xs:attribute name="name" type="xs:NMTOKEN" use="required"/>
182     <xs:attribute name="type" type="xs:QName" default="xs:string"/>
183     <xs:anyAttribute namespace="##other" processContents="lax"/>
184   </xs:complexType>
185 </xs:element>
186
187 <xs:element name="representation_variable">
188   <xs:complexType>
189     <xs:sequence>
190       <xs:any namespace="##other" processContents="lax"
191         minOccurs="0" maxOccurs="unbounded"/>
192     </xs:sequence>
193     <xs:attribute name="name" type="xs:NMTOKEN" use="required"/>
194     <xs:attribute name="type" type="xs:QName" default="xs:string"/>
195     <xs:attribute name="path" type="xs:string"/>
196     <xs:attribute name="required" type="xs:boolean" default="false"/>
197     <xs:attribute name="repeating" type="xs:boolean" default="false"/>
198     <xs:attribute name="fixed" type="xs:string"/>
199     <xs:anyAttribute namespace="##other" processContents="lax"/>
200   </xs:complexType>
201 </xs:element>
202
203 </xs:schema>

```

References

- [1] Yahoo! Web APIs. Technical report, Yahoo!, 2005. See <http://developer.yahoo.net/>.
- [2] Dave Raggett, Arnaud Le Hors, and Ian Jacobs. HTML (4.01 Specification). Recommendation, W3C, December 1999. See <http://www.w3.org/TR/html4/>.

Acknowledgments

Thanks to the members of the <http://lists.w3.org/Archives/Public/public-web-http-desc/> mailing list who provided useful feedback on several iterations of this specification. Mark Nottingham (BEA Systems) provided extensive feedback and helped structure the overall design.

Copyright Notice

Copyright 2005 Sun Microsystems, Inc., 4150 Network Circle, Santa Clara, California 95054, U.S.A. All rights reserved.

U.S. Government Rights - Commercial software. Government users are subject to the Sun Microsystems, Inc. standard license agreement and applicable provisions of the FAR and its supplements.

Sun, Sun Microsystems and the Sun logo are trademarks or registered trademarks of Sun Microsystems, Inc. in the U.S. and other countries.

DOCUMENTATION IS PROVIDED "AS IS" AND ALL EXPRESS OR IMPLIED CONDITIONS, REPRESENTATIONS AND WARRANTIES, INCLUDING ANY IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT, ARE DISCLAIMED, EXCEPT TO THE EXTENT THAT SUCH DISCLAIMERS ARE HELD TO BE LEGALLY INVALID.